

Amazon SageMaker と Ease of Development



丸山不二夫

Ease of Development について

EoD(Ease of Development)「開発の容易さ」と言うのは、開発環境とシステムの進化と開発者人口の拡大の、基本的なドライビング・フォースである。

Ease of Development について

- EoD(Ease of Development)「開発の容易さ」と言うのは、開発環境の進化の、基本的なドライビング・フォースである。
- ある場合には、逆に、「開発の難しさ」が、開発環境の整備には止まらず、肥大化したシステムのアーキテクチャーの見直し、へと進むこともある。これも、EoDの追求と考えていい。
- システムの進化とともに、一般には、システムの複雑さも増大するので、EoDの追求は、いつまでも終わることはない。ただ、EoDの追求なしには、いいシステムも、いい開発環境も生まれない。何よりも開発者の底辺の拡大にEoDは寄与する。
- 長い目で見れば、開発・運用環境としてのクラウドの利用、プログラミングでのIDEの利用、ソースコードの共有・管理でのGitHubの利用、CIの利用等、開発の容易さをめぐる環境は、大きく変化している。

Ease of Development について

- ソフトウェア的には、モジュールやコンポーネントを機能単位に構成して、その実装の詳細は隠して、インターフェースのみを利用者に公開すると言うのも、複雑化するシステムの中で、EoDを追求する有力な手法である。
- ディープラーニングのフレームワークでも、例えば、バックプロパゲーションのアルゴリズムを開発者はコードする必要はない。損失関数も、コスト最適化のアルゴリズムも、いくつかの候補を、フレームワークが提供する。開発者は、基本的には、その中から適当なものを選択する。それがフレームワークの役割である。
- システムや開発環境の問題としてだけでなく、開発者の問題として EoD を考えることができる。そこでは、モジュール/ツールを作る人とモジュール/ツールを使う人に、開発者の役割が分離することになる。それは、必要なことである。

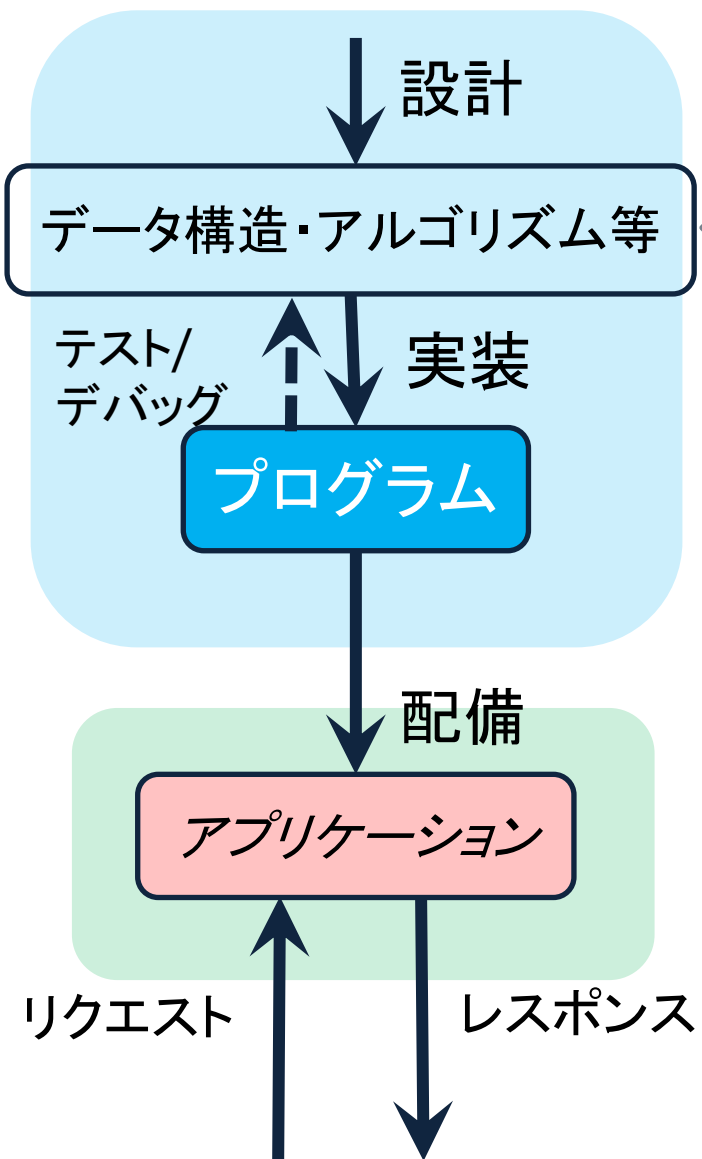
機械学習・深層学習でのサービス開発

機械学習・深層学習でのサービス開発は、従来のプログラミングによるサービス開発とは、大きく異なっている。

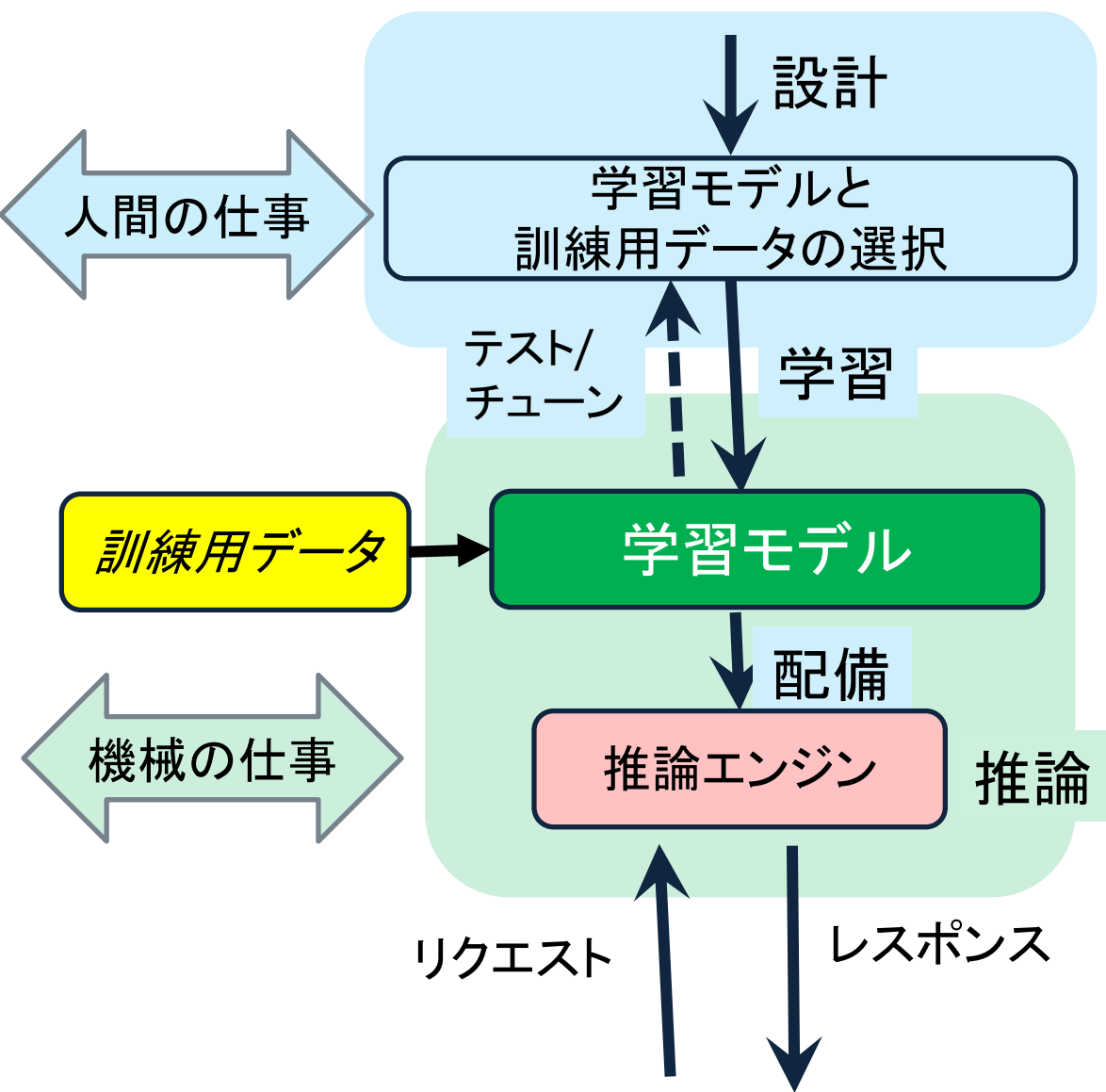
プログラミングによるサービス開発との比較

- 機械学習モデルは、基本的に統計学的モデルだが、ディープ・ラーニングのモデルは、ニューラル・ネットワーク・モデルを用いる。そうした違いはあるものの、「賢い」サービスを構成する開発のプロセスは、同じものである。
- このプロセスは、通常のプログラミング言語でサービスをプログラムする開発プロセスと比べると、人間が行う作業が圧倒的に少ない。人間が行っていたサービスのロジックをコードすることに相当する部分は、同じ働きをする推論エンジンの構築を、モデルの学習を通じて機械が代替する。(その分、CPU/GPUがフル稼働するのだが。)
- いいモデルが与えられて利用できるのなら、人間が行うことは、基本的には、モデルを選んで、いいデータを与えることが中心になる。

プログラミングによる サービス開発



機械学習・深層学習での サービス開発



データとモデルがポイントになる

□ データの生成と利用をめぐって

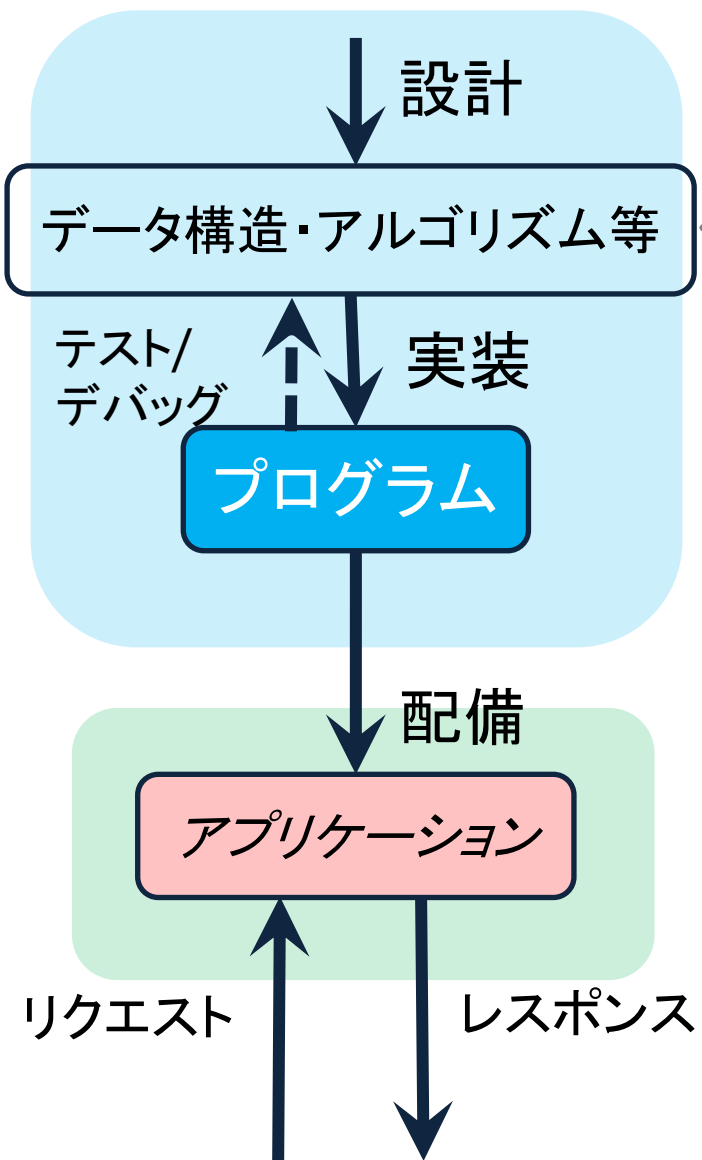
- ラベル付けへの人間の利用
AutoMLも！
- ラベルなし(教師なし)学習と強化学習
- 機械による自動生成
cf. AlphaZero (自分自身との対局データを利用)

□ モデルの生成と利用をめぐって

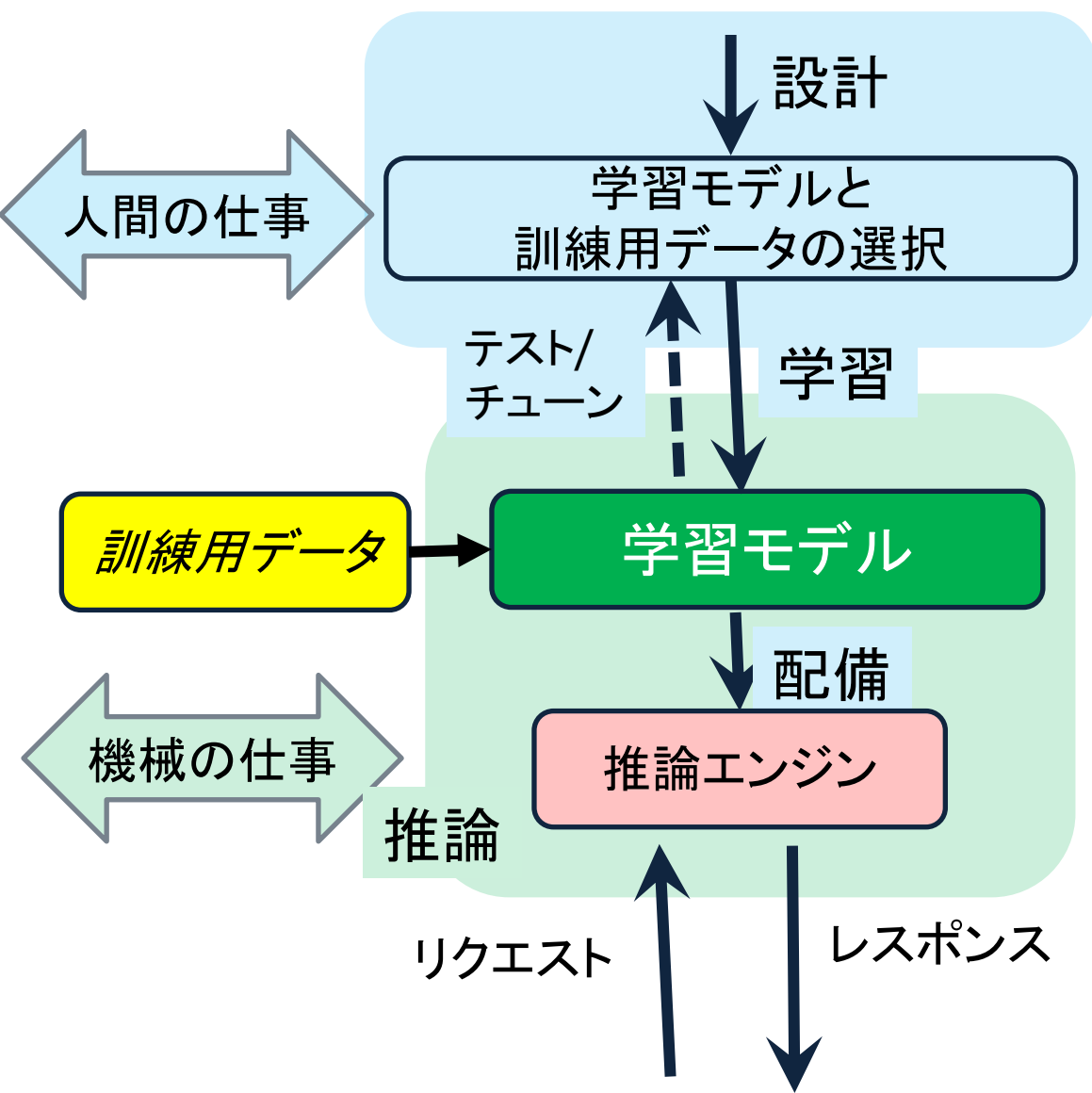
- モデル構築の共通API Gluon
- モデルの標準フォーマット ONNX, NNEF
- 転移学習
学習済みのモデルを、他の用途にも利用する。データの追加。
- モデルの自動生成 Learning to Learn (後述)

Amazon SageMakerでの開発の流れ

プログラミングによる サービス開発



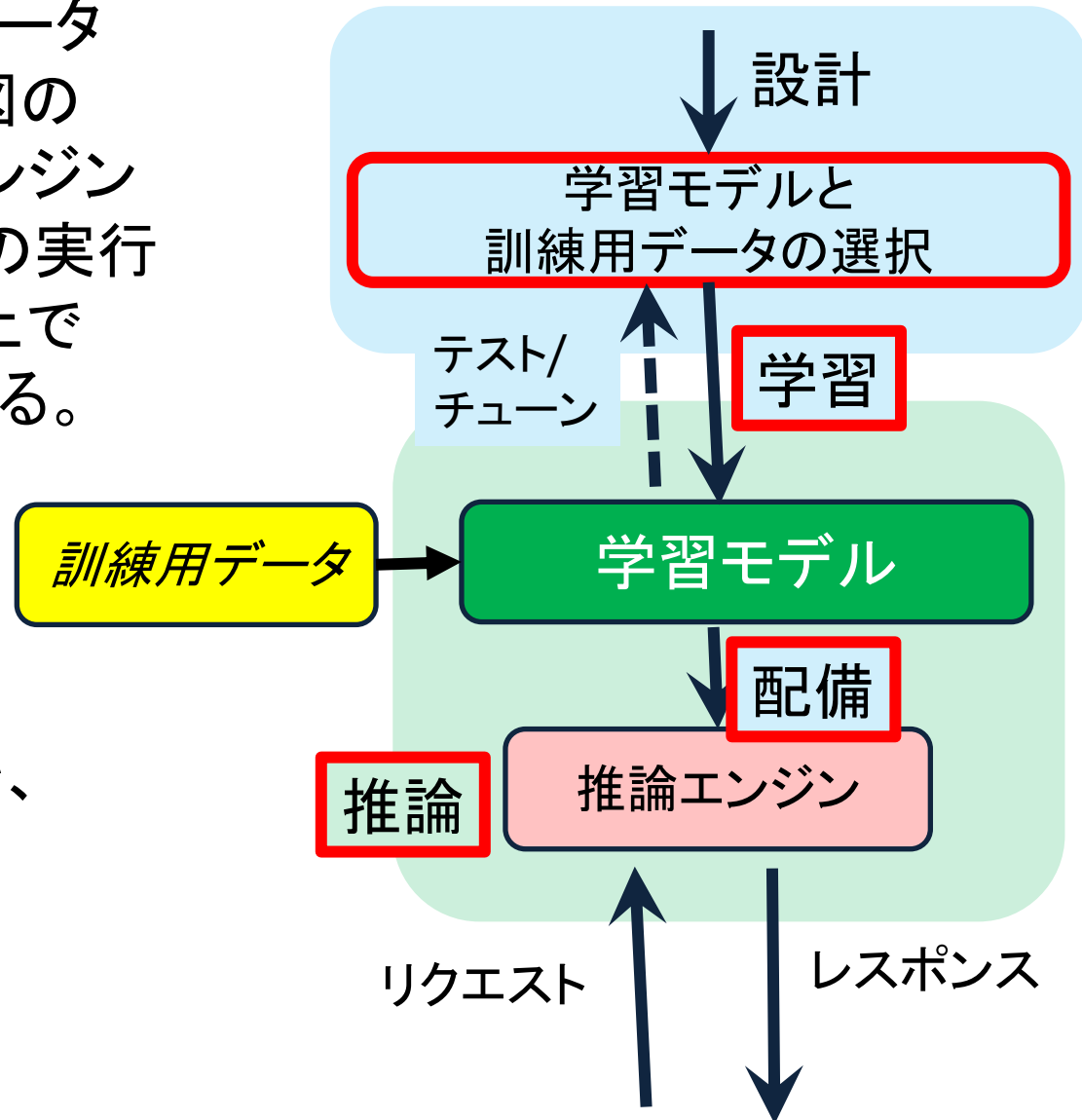
機械学習・深層学習での サービス開発



機械学習・深層学習での サービス開発

Amazon SageMakerは、
学習モデルと訓練用データ
の選択を行うと、右の図の
モデルの学習・推論エンジ
ンの配備・推論サービスの実行
までを、一つのツール上で
「一気通貫」で作り上げる。

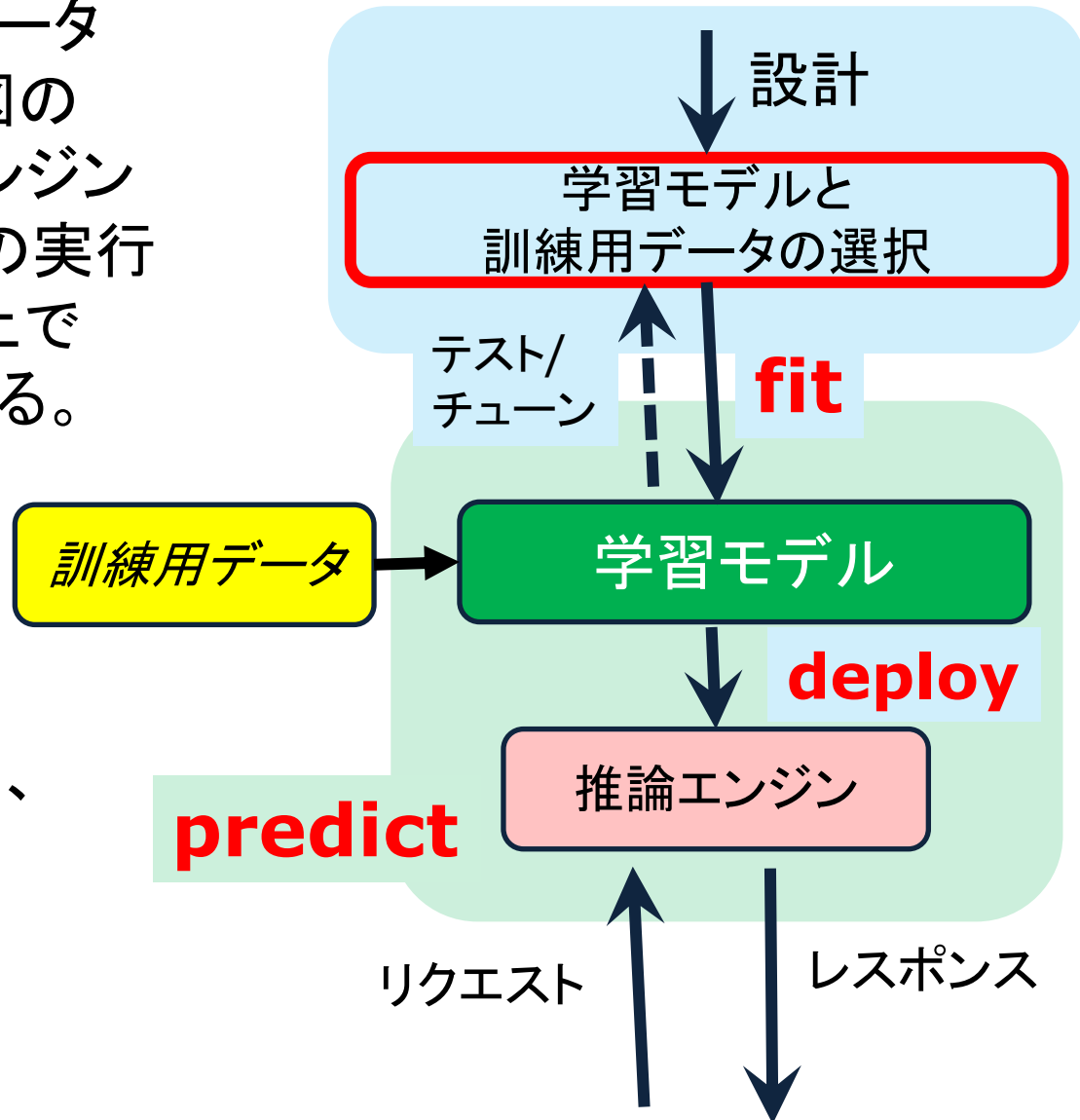
典型的な処理の流れを、
見ておこう。



機械学習・深層学習での サービス開発

Amazon SageMakerは、
学習モデルと訓練用データ
の選択を行うと、右の図の
モデルの学習・推論エンジ
ンの配備・推論サービスの実行
までを、一つのツール上で
「一気通貫」で作り上げる。

典型的な処理の流れを、
見ておこう。



モデルの生成

```
m = MXNet("mnist.py",  
  role=role,  
  train_instance_count=1,  
  train_instance_type="ml.c4.xlarge",  
  hyperparameters={'batch_size': 100,  
    'epochs': 20,  
    'learning_rate': 0.1,  
    'momentum': 0.9,  
    'log_interval': 100})
```

fit

```
m.fit(inputs)
```

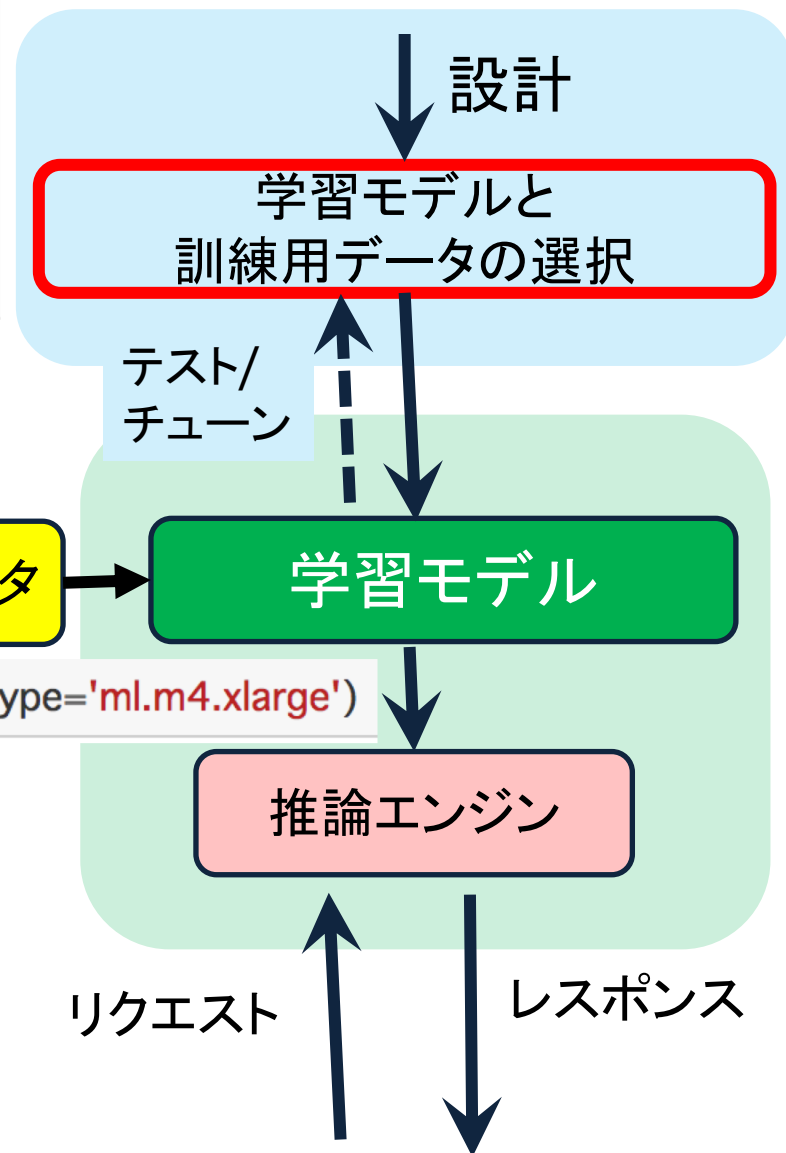
deploy

```
predictor = m.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

predict

```
response = predictor.predict(data)  
print int(response)
```

機械学習・深層学習での サービス開発



Amazon SageMakerでのサンプル

モデルmの生成

```
m = MXNet("mnist.py",  
          role=role,  
          train_instance_count=1,  
          train_instance_type="ml.c4.xlarge",  
          hyperparameters={'batch_size': 100,  
                           'epochs': 20,  
                           'learning_rate': 0.1,  
                           'momentum': 0.9,  
                           'log_interval': 100})
```

サンプル 1

**MXNetの
MNISTの
Gluonモデル**

<https://goo.gl/Sjsnc5>

モデルmの訓練: **fit**

```
m.fit(inputs)
```

推論エンジン predictorとして配備: **deploy**

```
predictor = m.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

推論の実行: **predict**

```
response = predictor.predict(data)  
print int(response)
```

モデルmの生成

```
m = MXNet("cifar10.py",
          role=role,
          train_instance_count=2,
          train_instance_type="ml.p2.xlarge",
          hyperparameters={'batch_size': 128,
                           'epochs': 50,
                           'learning_rate': 0.1,
                           'momentum': 0.9})
```

サンプル 2

**MXNetの
cifar10の
ResNetモデル**

<https://goo.gl/DQhQCP>

モデルmの訓練: **fit**

```
m.fit(inputs)
```

推論エンジン predictorとして配備: **deploy**

```
predictor = m.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

推論の実行: **predict**

```
for i, img in enumerate(image_data):
    response = predictor.predict(img)
    print('image {}: class: {}'.format(i, int(response)))
```

モデルmnist_estimatorの生成

```
mnist_estimator = TensorFlow(entry_point='mnist.py',  
                             role=role,  
                             training_steps=1000,  
                             evaluation_steps=100,  
                             train_instance_count=2,  
                             train_instance_type='ml.c4.xlarge')
```

サンプル 3

Tensorflowの MNIST 分散モデル

<https://goo.gl/otk9Cd>

モデルmnist_estimatorの訓練: **fit**

```
mnist_estimator.fit(inputs)
```

推論エンジン mnist_predictorとして配備: **deploy**

```
mnist_predictor = mnist_estimator.deploy(initial_instance_count=1,  
                                         instance_type='ml.m4.xlarge')
```

推論の実行: **predict**

```
predict_response = mnist_predictor.predict(tensor_proto)
```

モデルestimatorの生成

サンプル 4

```
estimator = TensorFlow(entry_point='resnet_cifar_10.py',
                        source_dir=source_dir,
                        role=role,
                        hyperparameters={'min_eval_frequency': 10},
                        training_steps=1000, evaluation_steps=100,
                        train_instance_count=2, train_instance_type='ml.c4.xlarge',
                        base_job_name='tensorboard-example')
```

Tensorflowの cifar10の ResNetモデル

<https://goo.gl/yRzhTE>

モデルestimatorの訓練: **fit**

```
estimator.fit(inputs, run_tensorboard_locally=True)
```

推論エンジン predictorとして配備: **deploy**

```
predictor = estimator.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

推論の実行: **predict**

```
response = predictor.predict(data)
print int(response)
```

SageMakerで、すぐに利用可能なモデル

SageMaker Built-in Algorithm

詳細は、リンクを参照されたい

- ❑ Linear Learner <https://goo.gl/4G9sV8>
- ❑ Factorization Machines <https://goo.gl/jJ9c3T>
- ❑ XGBoost Algorithm <https://goo.gl/GDnLgd>
- ❑ Image Classification Algorithm <https://goo.gl/nc3k2k>
- ❑ SageMaker Sequence2Sequence <https://goo.gl/McXLj2>
- ❑ K-Means Algorithm <https://goo.gl/V2HWiy>
- ❑ Principal Component Analysis <https://goo.gl/cXFGPF>
- ❑ Latent Dirichlet Allocation (LDA) <https://goo.gl/X6n54E>
- ❑ Neural Topic Model (NTM) <https://goo.gl/Fgbdhg>
- ❑ DeepAR Forecasting <https://goo.gl/gtsyQ6>
- ❑ BlazingText <https://goo.gl/Yktcm1>

<https://goo.gl/9xZdL4>


```
containers = {'us-west-2': '174872318107.dkr.ecr.us-west-2.amazonaws.com/linear-learner:latest',  
             'us-east-1': '382416733822.dkr.ecr.us-east-1.amazonaws.com/linear-learner:latest',  
             'us-east-2': '404615174143.dkr.ecr.us-east-2.amazonaws.com/linear-learner:latest',  
             'eu-west-1': '438346466558.dkr.ecr.eu-west-1.amazonaws.com/linear-learner:latest'}
```

```
import boto3  
import sagemaker  
  
sess = sagemaker.Session()  
  
linear = sagemaker.estimator.Estimator(containers[boto3.Session().region_name],  
                                       role,  
                                       train_instance_count=1,  
                                       train_instance_type='ml.c4.xlarge',  
                                       output_path=output_location,  
                                       sagemaker_session=sess)  
linear.set_hyperparameters(feature_dim=784,  
                           predictor_type='binary_classifier',  
                           mini_batch_size=200)  
  
linear.fit({'train': s3_train_data})
```

```
linear_predictor = linear.deploy(initial_instance_count=1,  
                                 instance_type='ml.m4.xlarge')
```

```
result = linear_predictor.predict(train_set[0][30:31])  
print(result)
```

**An Introduction to
Linear Learner
with MNIST**

<https://goo.gl/URsZqV>

```
containers = {'us-west-2': '174872318107.dkr.ecr.us-west-2.amazonaws.com/factorization-machines:latest',  
             'us-east-1': '382416733822.dkr.ecr.us-east-1.amazonaws.com/factorization-machines:latest',  
             'us-east-2': '404615174143.dkr.ecr.us-east-2.amazonaws.com/factorization-machines:latest',  
             'eu-west-1': '438346466558.dkr.ecr.eu-west-1.amazonaws.com/factorization-machines:latest'}
```

```
import boto3  
import sagemaker  
  
sess = sagemaker.Session()  
  
fm = sagemaker.estimator.Estimator(containers[boto3.Session().region_name],  
                                   role,  
                                   train_instance_count=1,  
                                   train_instance_type='ml.c4.xlarge',  
                                   output_path=output_location,  
                                   sagemaker_session=sess)  
fm.set_hyperparameters(feature_dim=784,  
                       predictor_type='binary_classifier',  
                       mini_batch_size=200,  
                       num_factors=10)  
  
fm.fit({'train': s3_train_data})
```

```
fm_predictor = fm.deploy(initial_instance_count=1,  
                          instance_type='ml.m4.xlarge')
```

```
result = fm_predictor.predict(train_set[0][30:31])  
print(result)
```

**An Introduction to
Factorization Machines
with MNIST**

<https://goo.gl/u2UDvq>

```
# select the algorithm container based on this notebook's current location
```

```
containers = {  
    'us-west-2': '266724342769.dkr.ecr.us-west-2.amazonaws.com/lda:latest',  
    'us-east-1': '766337827248.dkr.ecr.us-east-1.amazonaws.com/lda:latest',  
    'us-east-2': '999911452149.dkr.ecr.us-east-2.amazonaws.com/lda:latest',  
    'eu-west-1': '999678624901.dkr.ecr.eu-west-1.amazonaws.com/lda:latest'  
}  
region_name = boto3.Session().region_name  
container = containers[region_name]  
  
print('Using SageMaker LDA container: {} ({}).format(container, region_name))
```

```
session = sagemaker.Session()
```

```
# specify general training job information
```

```
lda = sagemaker.estimator.Estimator(  
    container,  
    role,  
    output_path='s3://{}/{}'/output'.format(bucket, prefix),  
    train_instance_count=1,  
    train_instance_type='ml.c4.2xlarge',  
    sagemaker_session=session,  
)
```

**An Introduction to
SageMaker LDA**

```
# set algorithm-specific hyperparameters
lda.set_hyperparameters(
    num_topics=num_topics,
    feature_dim=vocabulary_size,
    mini_batch_size=num_documents_training,
    alpha0=1.0,
)

# run the training job on input data stored in S3
lda.fit({'train': s3_train_data})
```

```
lda_inference = lda.deploy(
    initial_instance_count=1,
    instance_type='ml.m4.xlarge', # LDA inference may work better at scale on ml.c4 instances
)
```

```
results = lda_inference.predict(documents_test[:12])
print(results)
```

**An Introduction to
SageMaker LDA**

利用可能な、訓練済みのモデル

MXNet Model Zoo (訓練済みのモデル有り)

□ CNN

- [ImageNet](#)
- [CIFAR10](#)
- [PASCAL VOC](#)
- [UCF101](#)
- [Places2](#)
- [Multimedia Commons](#)

□ RNN

- [Penn Treebank \(PTB\)](#)
- [Shakespeare](#)
- [IMDB reviews](#)
- [Facebook bAbI](#)



...

https://mxnet.apache.org/model_zoo/index.html

Gluon Model Zoo (訓練済みのモデル有り)

- ❑ [AlexNet](#)
- ❑ [DenseNet](#)
- ❑ [Inception V3](#)
- ❑ [ResNet V1](#)
- ❑ [ResNet V2](#)
- ❑ [SqueezeNet](#)
- ❑ [VGG](#)
- ❑ [MobileNet](#)
- ❑ [MobileNetV2](#)

Tensorflow Estimator

https://www.tensorflow.org/programmers_guide/estimators

“Tensorflow, Machine Learning and Learning to learn” Sherry Moore

<https://goo.gl/Gr2jvX>

Canned Estimators

Models in a box

Estimator

tf.keras

Build models

tf.layers

Python Frontend

C++ Frontend

...

TensorFlow Distributed Execution Engine

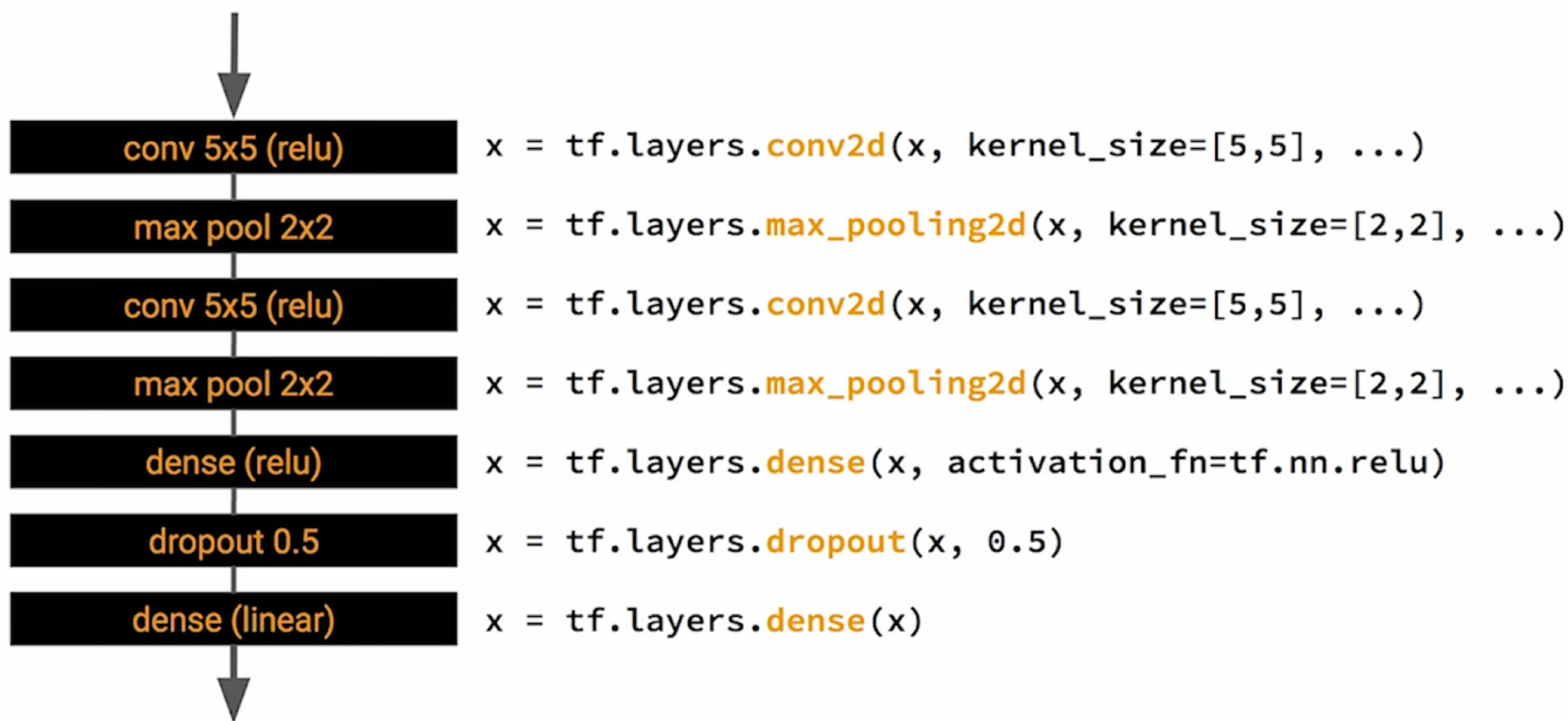
CPU

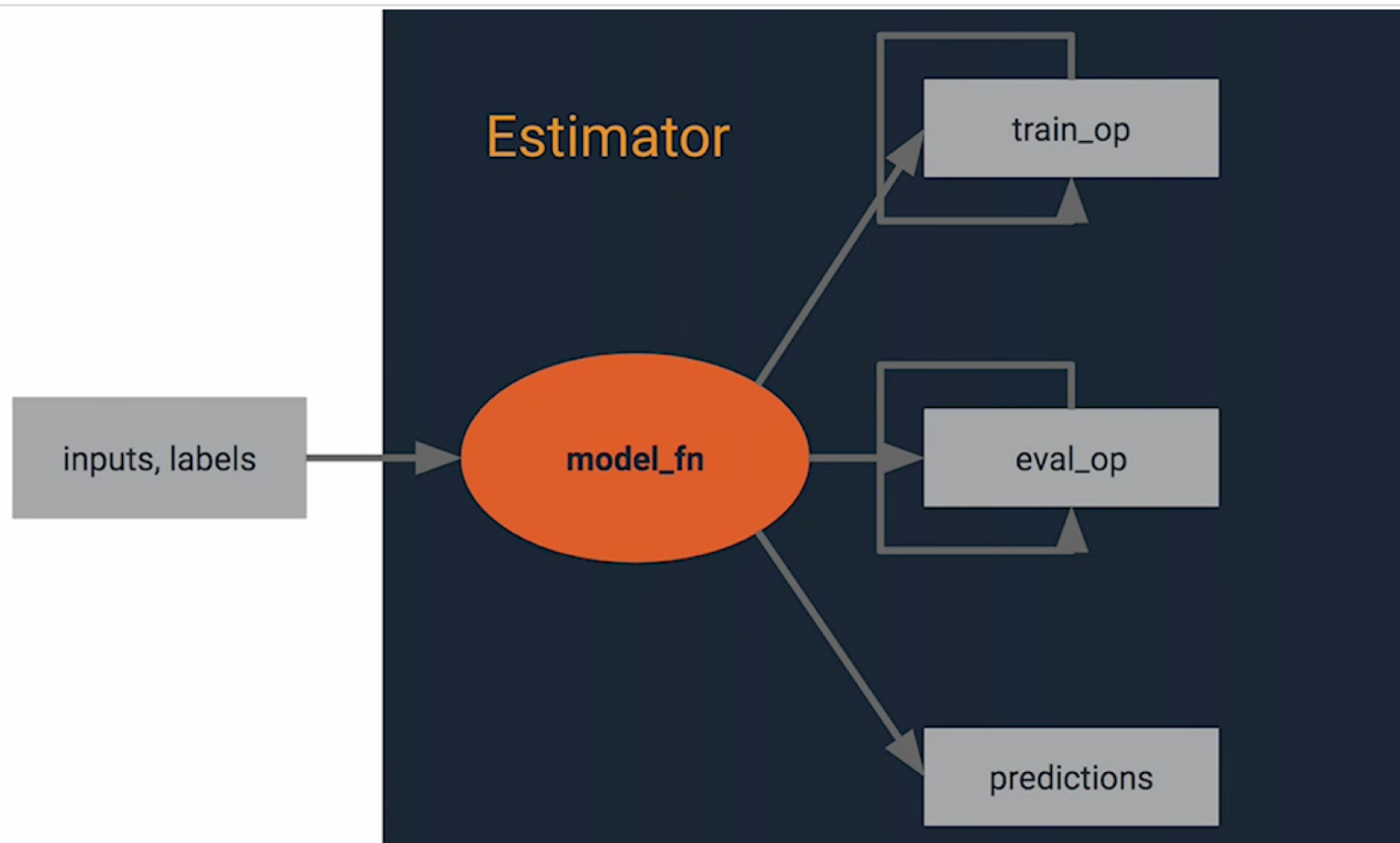
GPU

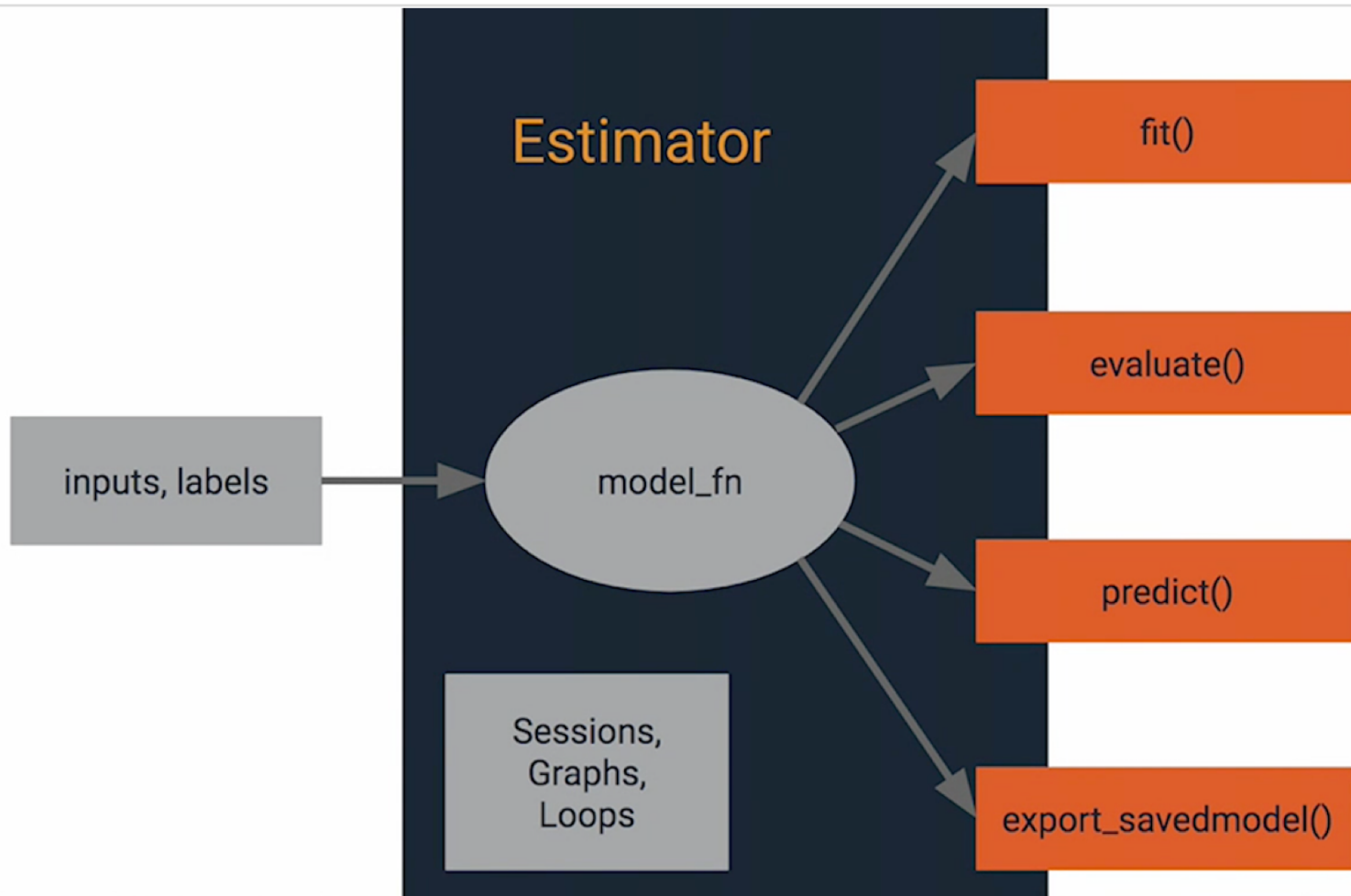
Android

iOS

...



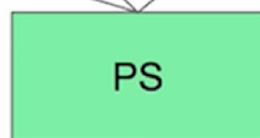
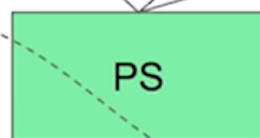
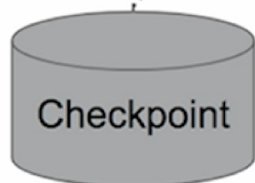
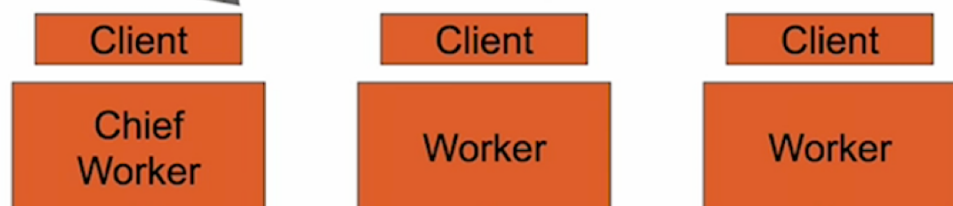
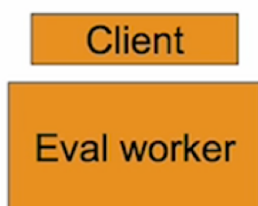




```
area = real_valued_column("square_foot"),  
rooms = real_valued_column("num_rooms"),  
zip_code = sparse_column_with_integerized_feature("zip_code", 100000)  
  
regressor = LinearRegressor(feature_columns=[area, rooms, zip_code])  
  
classifier.fit(train_input_fn)  
classifier.evaluate(eval_input_fn)
```

estimator.evaluate()

estimator.train()



start TensorFlow server

```
TF_CONFIG = {  
    "cluster": {  
        "chief": ["chief.example.com:2222"],  
        "worker": ["worker0.example.com:2222",  
                   "worker1.example.com:2222",  
                   "worker2.example.com:2222"],  
        "ps": ["ps0.example.com:2222",  
               "ps1.example.com:2222"]  
    },  
    "task": {"type": "worker", "index": 1},  
    "model_dir": "/path/to/model",  
}
```

```
# Create an Estimator
```

```
est = tf.estimator.Estimator(...)
```

```
# Start training, eval, or PS
```

```
tf.estimator.train_and_evaluate(
```

```
    est,
```

```
    training_input_fn=..., # Specify training inputs
```

```
    [EvalSpec(eval_input_fn, ...)]) # Specify eval configuration
```

```
# This code can run on all nodes
```

```
# Configured by environment TF_CONFIG
```

AIでのEoDの取り組みのもう一つの例

Google AutoML

AIでのEoD

Google AutoMLの背景

- その一方で、現在、機械学習や AI の進歩を十分に理解して活用できる人材やリソースを持っている企業は、世界でもほんの一部に限られています。高度な機械学習モデルを開発できる人材に於いては、その数はさらに限られるでしょう。さらに、機械学習や AI エンジニアの力を活用できたとしても、独自の機械学習モデルを構築するには時間がかかり、複雑なプロセス管理が必要です。Google は、特定のタスクを実行するための 学習済みの機械学習モデル を提供していますが、あらゆる人が AI を活用できるようにする、という目標のためには、さらなる努力が必要です。

AIでのEoD

Google AutoML

- 本日、Google では Cloud AutoML を発表します。Cloud AutoML は、機械学習や AI のリソースが限られた環境でも、[learning2learn](#) や[転移学習](#)のような高度な技術を活用し、独自の高品質でカスタマイズされた機械学習モデルの構築を可能にします。十分な人材やリソースを持たない企業においてもパワフルな AI システムを構築でき、さらには、AI エキスパートの生産性を高めたり、AI 活用の幅を広げるツールとなることを期待しています。

Cloud AutoML Vision

Train Custom Machine Learning Models

Cloud AutoML is a suite of Machine Learning products that enables developers with limited machine learning expertise to train high quality models by leveraging Google's state of the art transfer learning, and Neural Architecture Search technology.



Generate high quality training data

You can use Google's human labeling service to annotate or clean your image labels to make sure your models are being trained on high-quality data.

~~Mannequin~~

~~Statue~~

Human

~~Dog~~

~~Bench~~



<https://cloud.google.com/automl/>

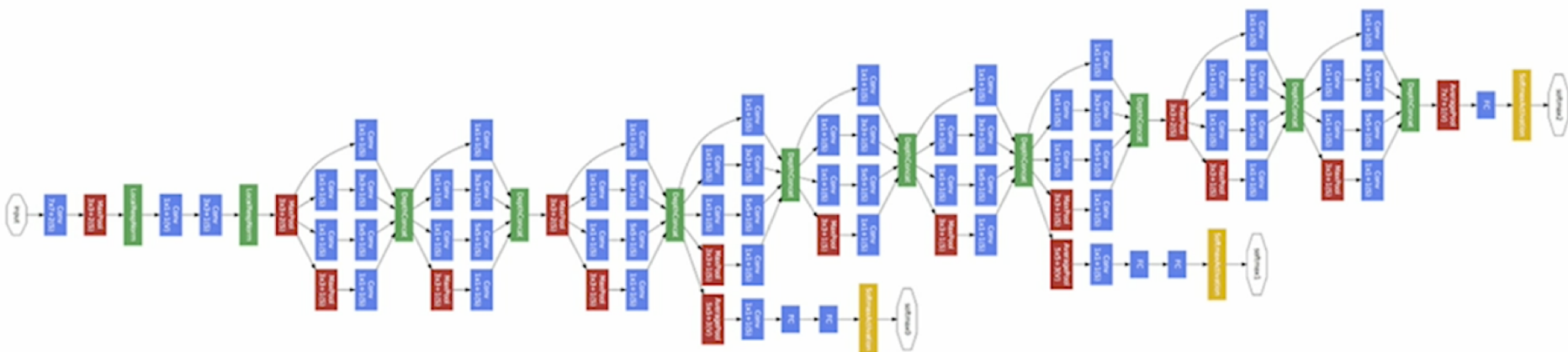
Learning to Learn

“Tensorflow, Machine Learning and Learning to learn” Sherry Moore

<https://goo.gl/Gr2jvX>

モデルの自動生成 Learn2Learn

画像認識のInception Model



Rethinking the Inception Architecture for Computer Vision

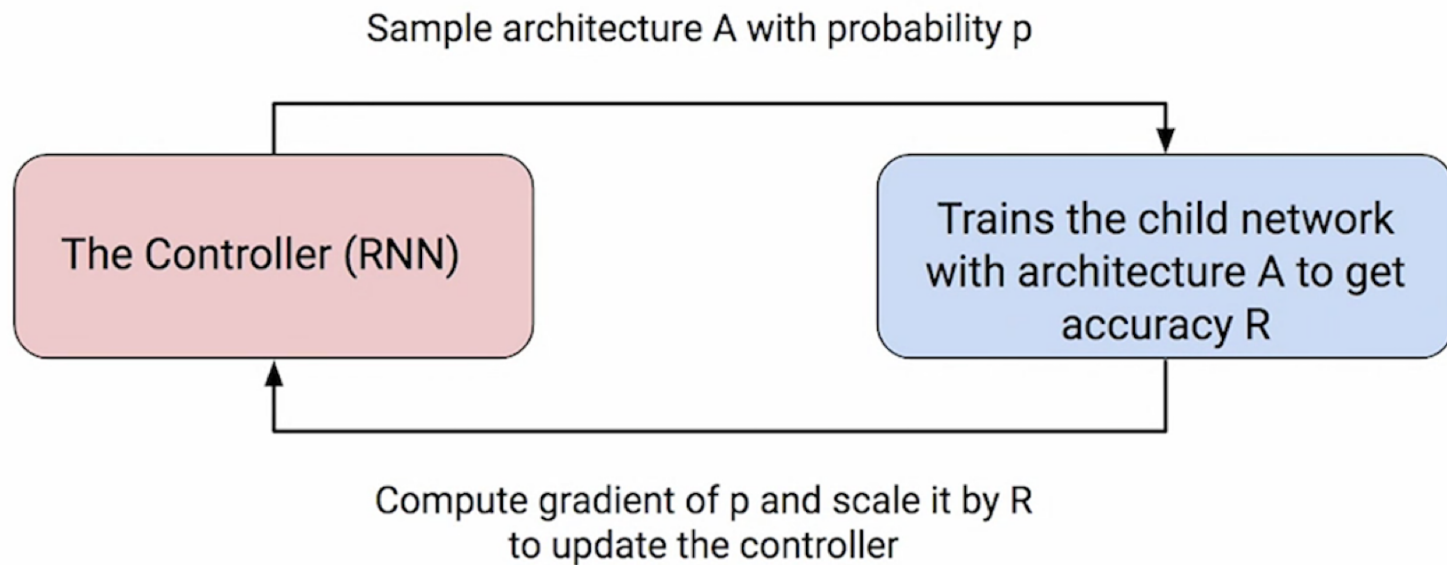
Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna





#OReillyAI

OREILLY
Artificial
Intelligence



*Neural Architecture Search using
Reinforcement Learning,*
Barret Zoph and Quoc Le.

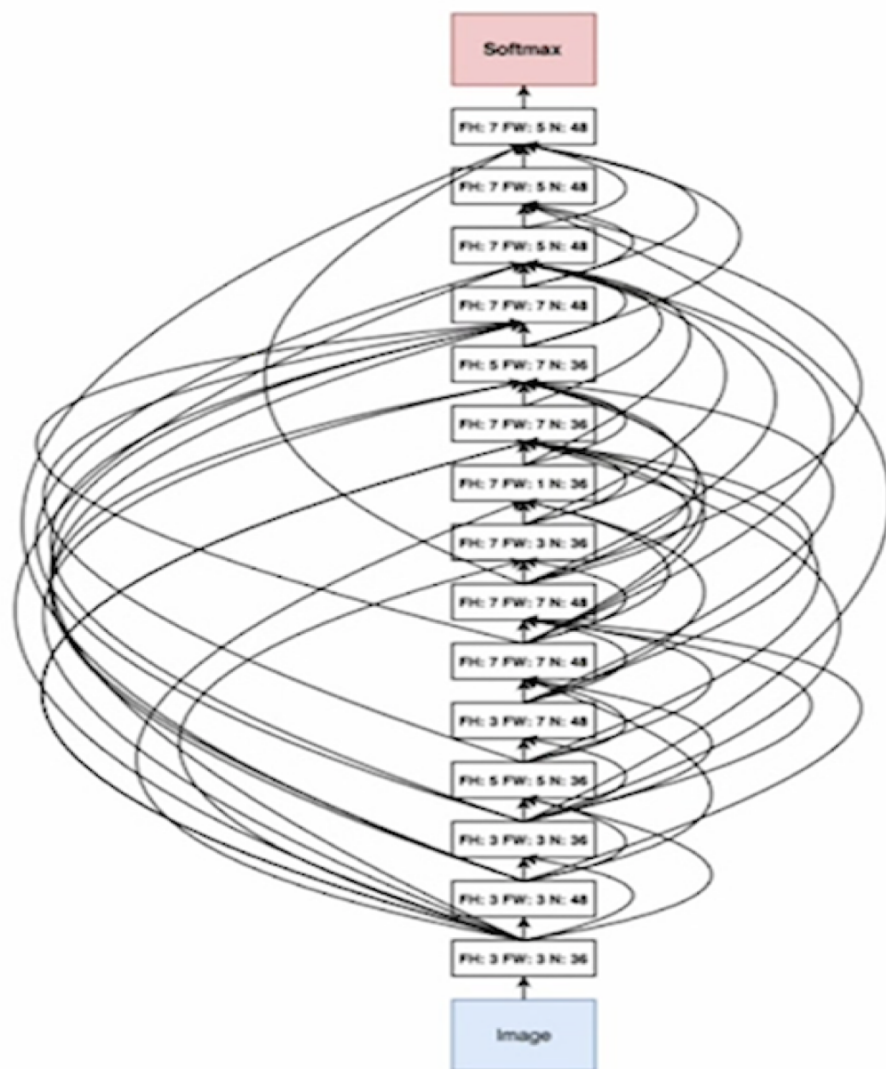
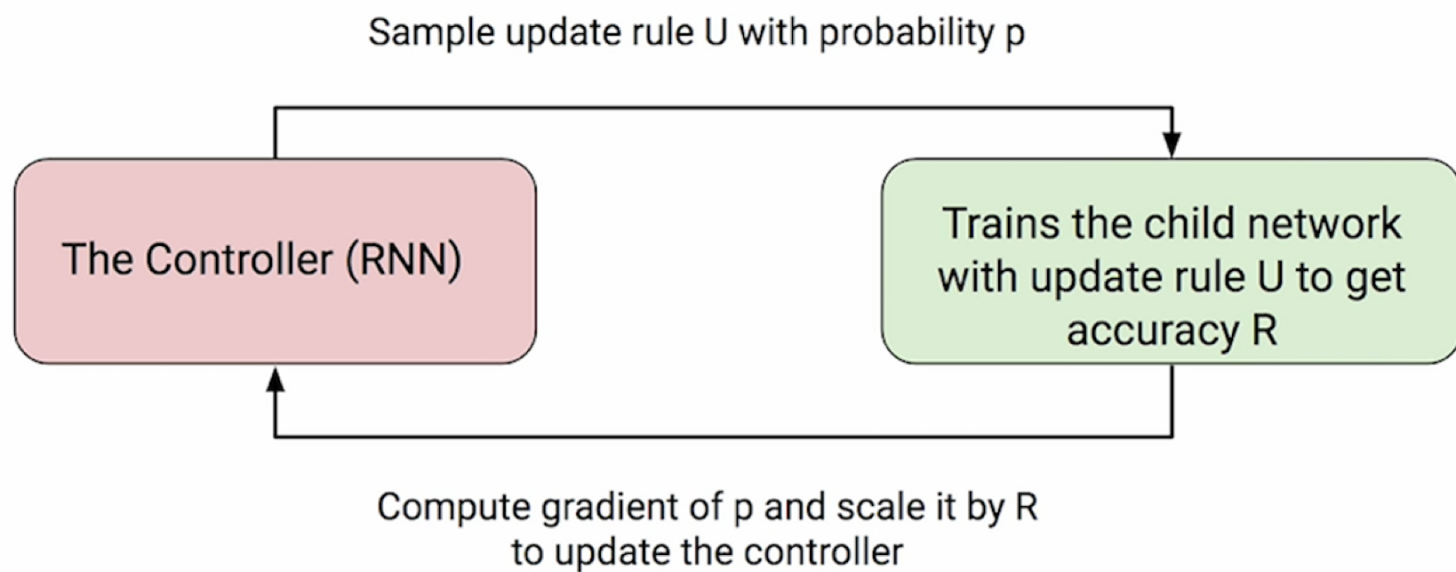


Figure 7: Convolutional architecture discovered by our method, when the search space does not have strides or pooling layers. FH is filter height, FW is filter width and N is number of filters.



*Neural Optimizer Search using
Reinforcement Learning,*
Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc Le.

Optimizer	Final Val	Final Test	Best Val	Best Test
SGD	92.0	91.8	92.9	91.9
Momentum	92.7	92.1	93.1	92.3
ADAM	90.4	90.1	91.8	90.7
RMSProp	90.7	90.3	91.4	90.3
$[e^{\text{sign}(g) * \text{sign}(m)} + \text{clip}(g, 10^{-4})] * g$	92.5	92.4	93.8	93.1
$\text{clip}(\hat{m}, 10^{-4}) * e^{\hat{v}}$	93.5	92.5	93.8	92.7
$\hat{m} * e^{\hat{v}}$	93.1	92.4	93.8	92.6
$g * e^{\text{sign}(g) * \text{sign}(m)}$	93.1	92.8	93.8	92.8
$\text{drop}(g, 0.3) * e^{\text{sign}(g) * \text{sign}(m)}$	92.7	92.2	93.6	92.7
$\hat{m} * e^{g^2}$	93.1	92.5	93.6	92.4
$\text{drop}(\hat{m}, 0.1) / (e^{g^2} + \epsilon)$	92.6	92.4	93.5	93.0
$\text{drop}(g, 0.1) * e^{\text{sign}(g) * \text{sign}(m)}$	92.8	92.4	93.5	92.2
$\text{clip}(\text{RMSProp}, 10^{-5}) + \text{drop}(\hat{m}, 0.3)$	90.8	90.8	91.4	90.9
$\text{ADAM} * e^{\text{sign}(g) * \text{sign}(m)}$	92.6	92.0	93.4	92.0
$\text{ADAM} * e^{\hat{m}}$	92.9	92.8	93.3	92.7
$g + \text{drop}(\hat{m}, 0.3)$	93.4	92.9	93.7	92.9
$\text{drop}(\hat{m}, 0.1) * e^{g^3}$	92.8	92.7	93.7	92.8
$g - \text{clip}(g^2, 10^{-4})$	93.4	92.8	93.7	92.8
$e^g - e^{\hat{m}}$	93.2	92.5	93.5	93.1
$\text{drop}(\hat{m}, 0.3) * e^w$	93.2	93.0	93.5	93.2

Table 1. Performance of Neural Optimizer Search and standard optimizers on the Wide-ResNet architecture (Zagoruyko & Komodakis, 2016) on CIFAR-10. Final Val and Final Test refer to the final validation and test accuracy after for training for 300 epochs. Best Val corresponds to the best validation accuracy over the 300 epochs and Best Test is the test accuracy at the epoch where the validation accuracy was the highest.

Large-Scale Evolution of Image Classifiers

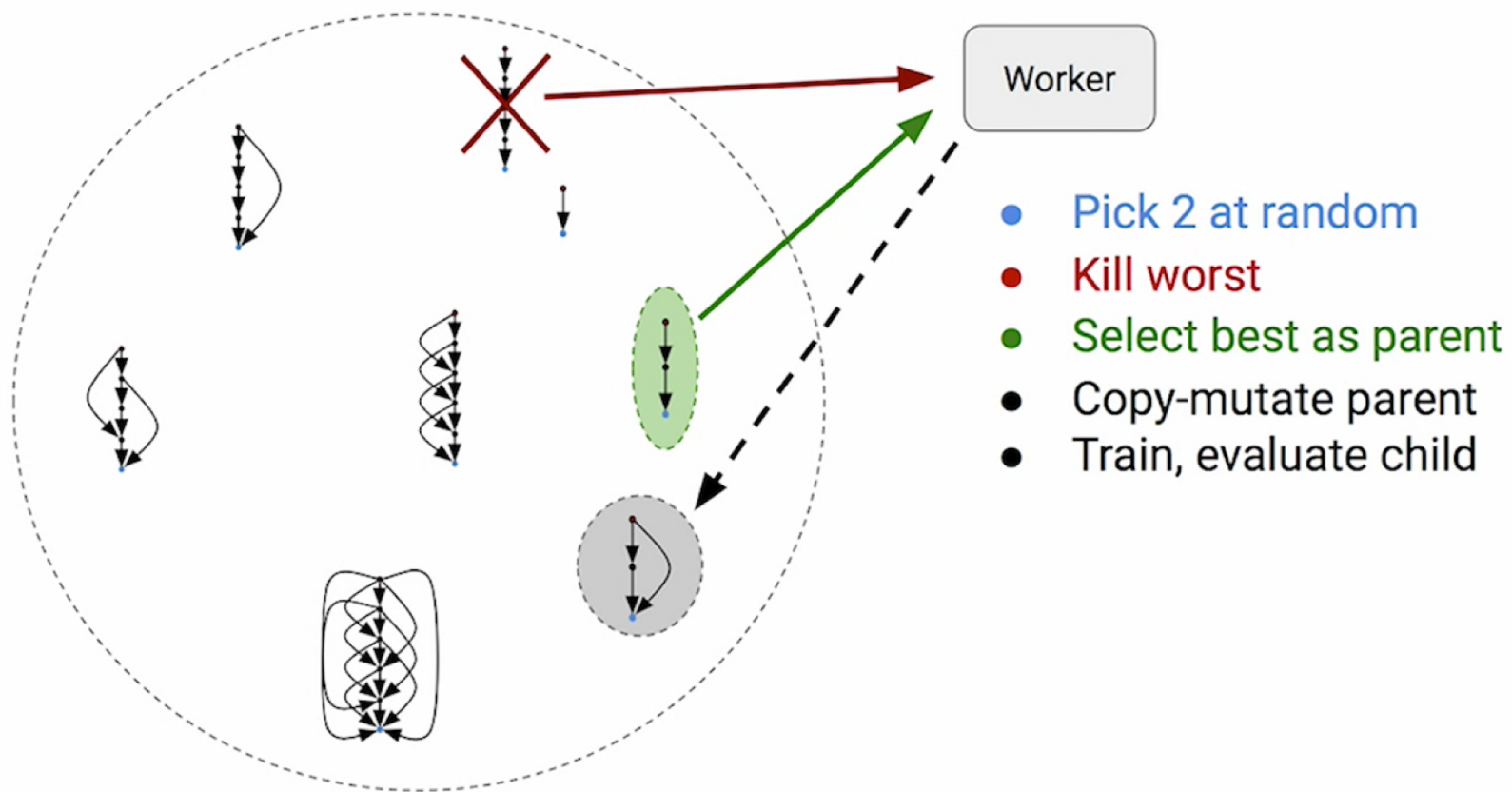
Google

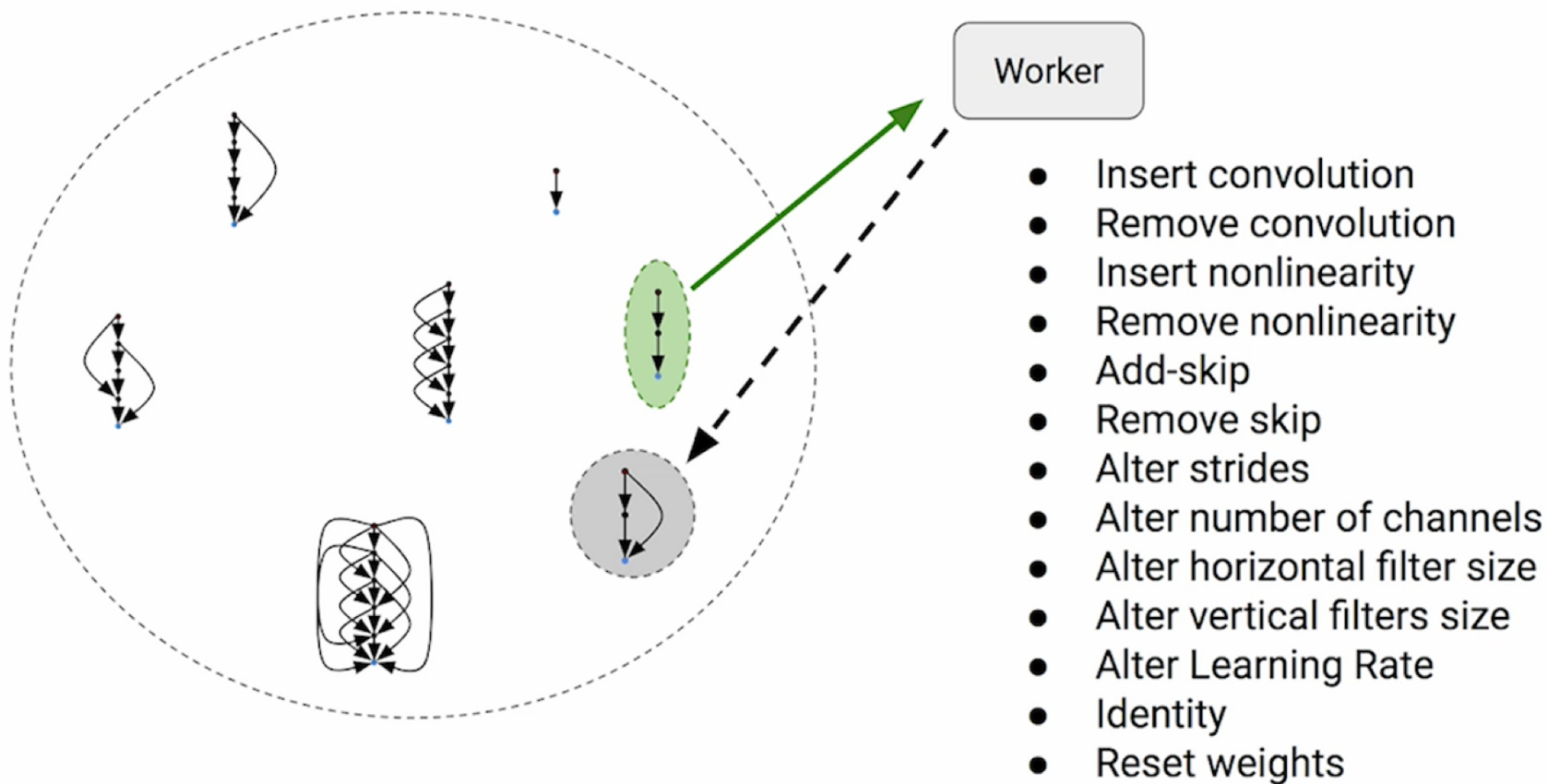


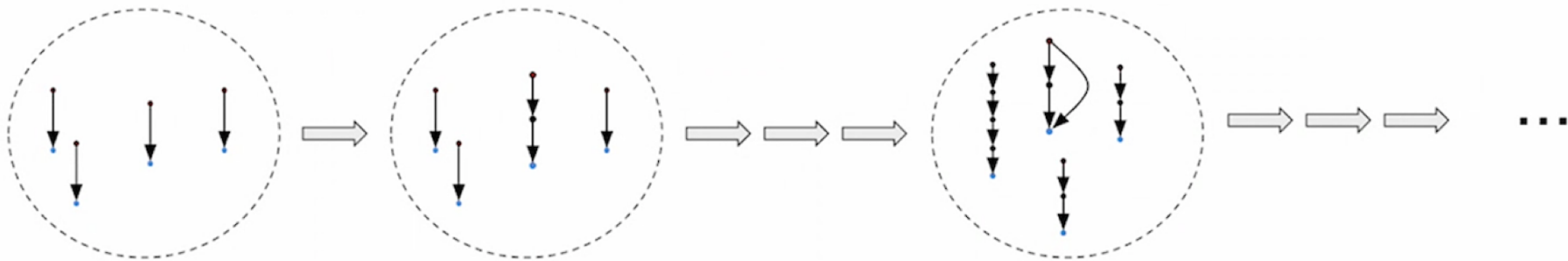
Esteban Real, Sherry Moore, Andrew Selle,
Saurabh Saxena, Yutaka Leon Suematsu,
Jie Tan, Quoc Le, Alex Kurakin

Google

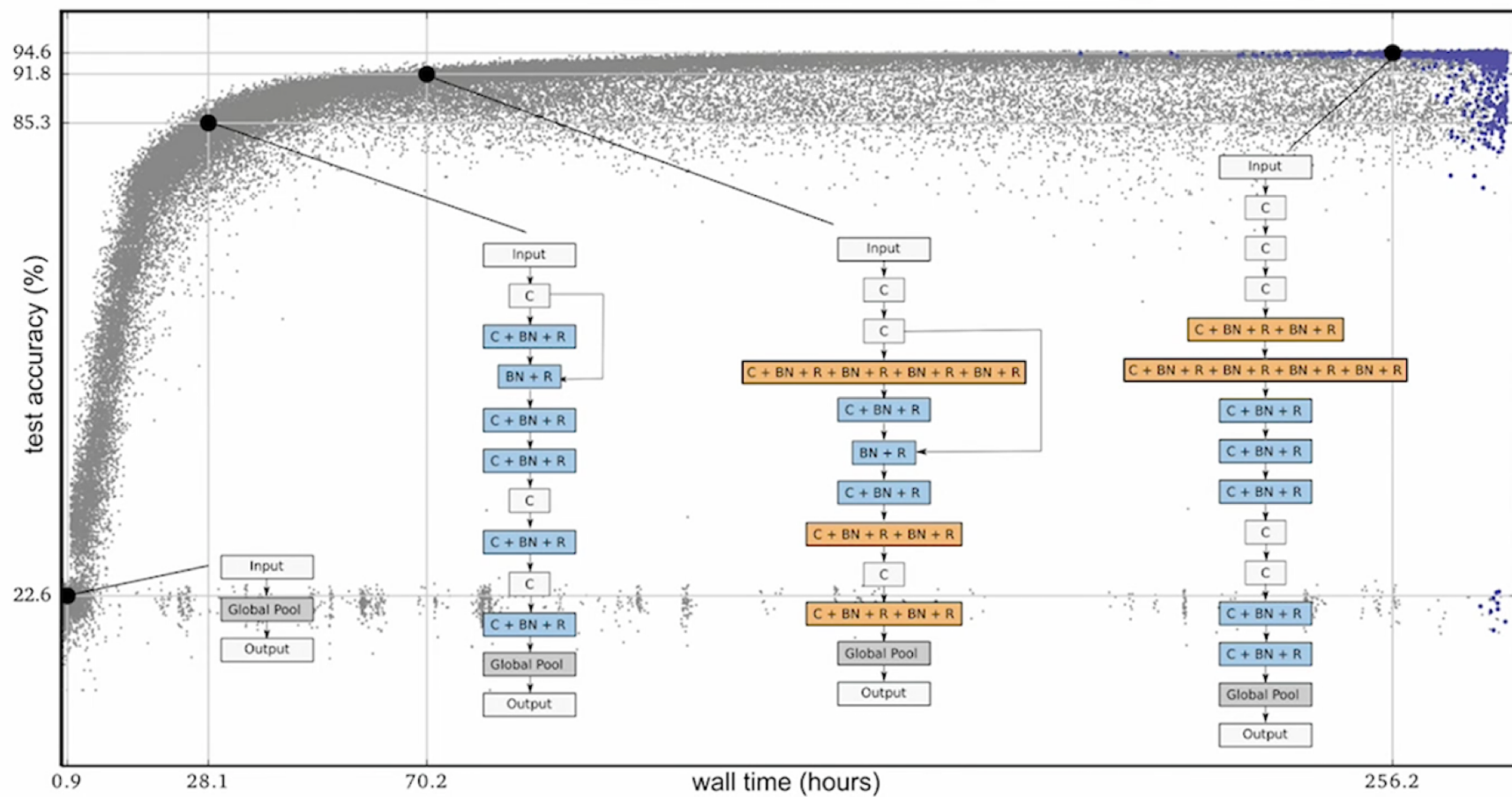








- Initialize with trivial models
 - Repeat evolutionary step
 - Get fully trained model
-



Compared with Hand-designed Models

STUDY	PARAMS.	C10+	C100+	WITHIN?
MAXOUT (GOODFELLOW ET AL., 2013)	–	90.7%	61.4%	No
NETWORK IN NETWORK (LIN ET AL., 2013)	–	91.2%	–	No
ALL-CNN (SPRINGENBERG ET AL., 2014)	1.3 M	92.8%	66.3%	YES
DEEPLY SUPERVISED (LEE ET AL., 2015)	–	92.0%	65.4%	No
HIGHWAY (SRIVASTAVA ET AL., 2015)	2.3 M	92.3%	67.6%	No
RESNET (HE ET AL., 2016)	1.7 M	93.4%	72.8% [†]	YES
EVOLUTION (OURS)	5.4 M 40.4 M	94.6%	76.0%	N/A
WIDE RESNET 28-10 (ZAGORUYKO & KOMODAKIS, 2016)	36.5 M	96.0%	80.0%	YES
WIDE RESNET 40-10+D/O (ZAGORUYKO & KOMODAKIS, 2016)	50.7 M	96.2%	81.7%	No
DENSENET (HUANG ET AL., 2016A)	25.6 M	96.7%	82.8%	No

Comparison with Other Architecture Search Methods

STUDY	STARTING POINT	CONSTRAINTS	POST-PROCESSING	PARAMS.	C10+	C100+
BAYESIAN (SNOEK ET AL., 2012)	3 LAYERS	FIXED ARCHITECTURE, NO SKIPS	NONE	–	90.5%	–
Q-LEARNING (BAKER ET AL., 2016)	–	DISCRETE PARAMS., MAX. NUM. LAYERS, NO SKIPS	TUNE, RETRAIN	11.2 M	93.1%	72.9%
RL (ZOPH & LE, 2016)	20 LAYERS, 50% SKIPS	DISCRETE PARAMS., EXACTLY 20 LAYERS	SMALL GRID SEARCH, RETRAIN	2.5 M	94.0%	–
RL (ZOPH & LE, 2016)	39 LAYERS, 2 POOL LAYERS AT 13 AND 26, 50% SKIPS	DISCRETE PARAMS., EXACTLY 39 LAYERS, 2 POOL LAYERS AT 13 AND 26	ADD MORE FILTERS, SMALL GRID SEARCH, RETRAIN	37.0 M	96.4%	–
EVOLUTION (OURS)	SINGLE LAYER, ZERO CONVS.	POWER-OF-2 STRIDES	NONE	5.4 M 40.4 M ENSEMB.	94.6% 95.6%	77.0%

A white, cube-shaped smart device with a camera on top. The front face features the AWS logo, an Intel logo, and a circular sensor. The background is a blurred indoor setting.

いつか、誰でも、人工知能を
簡単に作れる時代が来る。

Amazon SageMaker
クラウド・ハンズオン